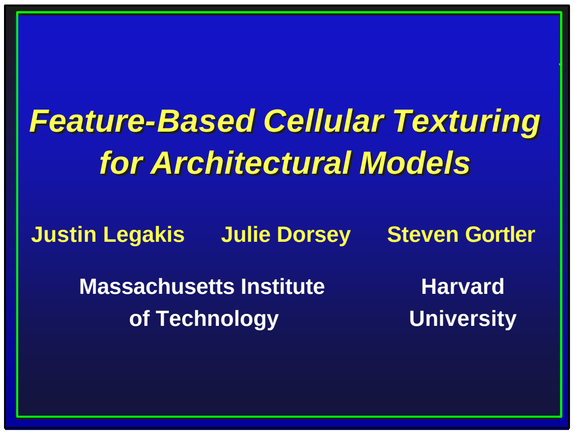
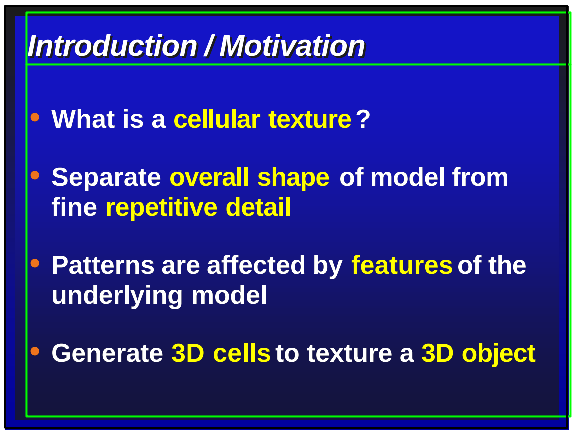


Feature-Based Cellular Texturing for Architectural Models

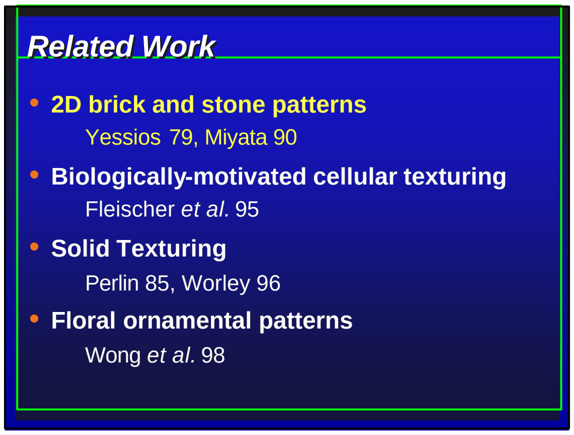
Justin Legakis   Julie Dorsey   Steven Gortler

Massachusetts Institute of Technology   Harvard University



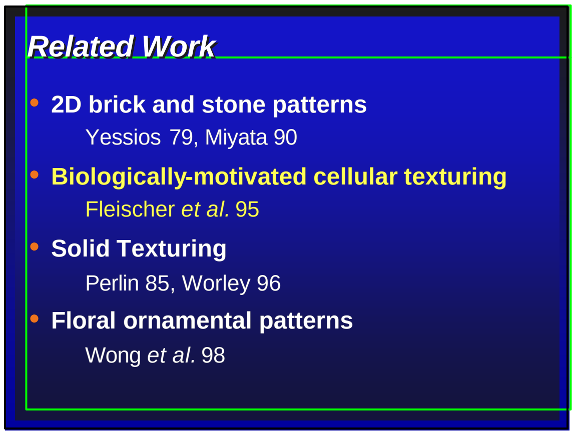
Introduction / Motivation

- What is a cellular texture?
- Separate overall shape of model from fine repetitive detail
- Patterns are affected by features of the underlying model
- Generate 3D cells to texture a 3D object



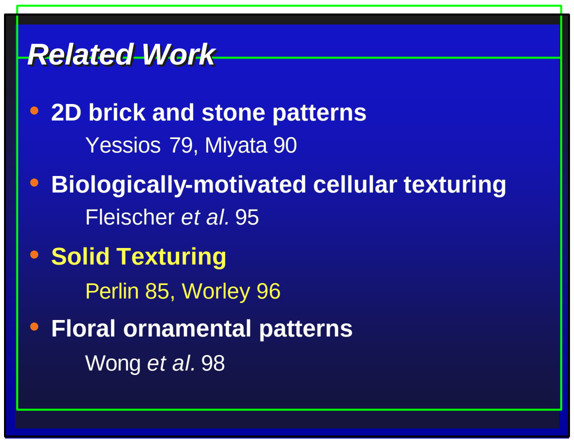
Related Work

- 2D brick and stone patterns
  Yessios 79, Miyata 90
- Biologically-motivated cellular texturing
  Fleischer *et al.* 95
- Solid Texturing
  Perlin 85, Worley 96
- Floral ornamental patterns
  Wong *et al.* 98



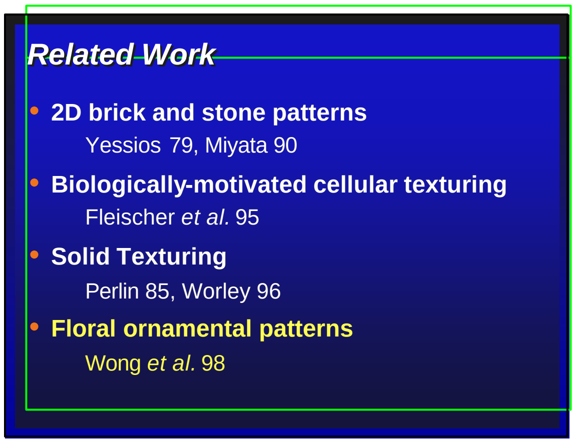
Related Work

- 2D brick and stone patterns
  Yessios 79, Miyata 90
- Biologically-motivated cellular texturing
  Fleischer *et al.* 95
- Solid Texturing
  Perlin 85, Worley 96
- Floral ornamental patterns
  Wong *et al.* 98



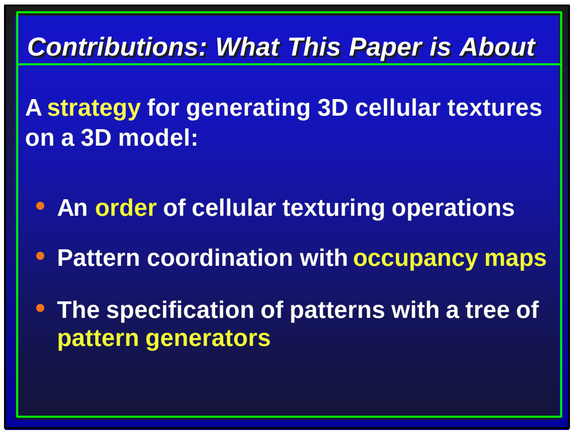
Related Work

- 2D brick and stone patterns
  Yessios 79, Miyata 90
- Biologically-motivated cellular texturing
  Fleischer *et al.* 95
- Solid Texturing
  Perlin 85, Worley 96
- Floral ornamental patterns
  Wong *et al.* 98



Related Work

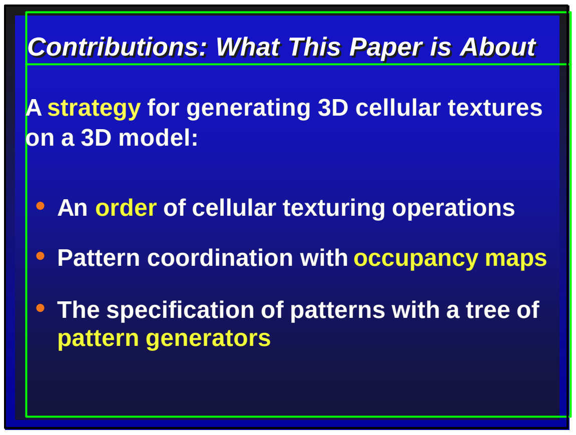
- 2D brick and stone patterns
  Yessios 79, Miyata 90
- Biologically-motivated cellular texturing
  Fleischer *et al.* 95
- Solid Texturing
  Perlin 85, Worley 96
- Floral ornamental patterns
  Wong *et al.* 98



Contributions: What This Paper is About

A strategy for generating 3D cellular textures on a 3D model:

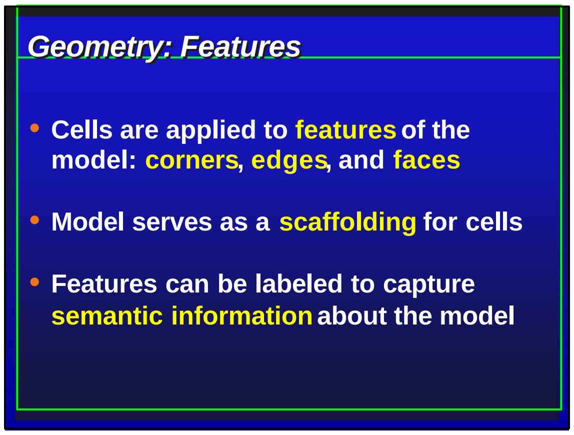
- An order of cellular texturing operations
- Pattern coordination with occupancy maps
- The specification of patterns with a tree of pattern generators



Contributions: What This Paper is About

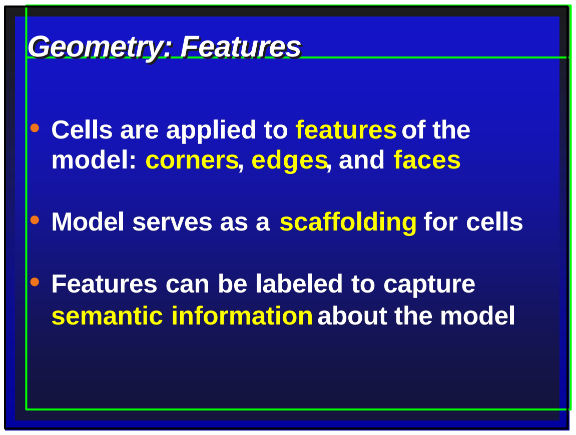
A strategy for generating 3D cellular textures on a 3D model:

- An order of cellular texturing operations
- Pattern coordination with occupancy maps
- The specification of patterns with a tree of pattern generators



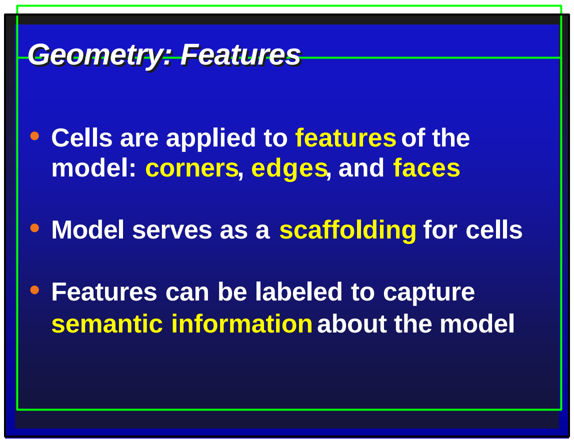
Geometry: Features

- Cells are applied to features of the model: corners, edges, and faces
- Model serves as a scaffolding for cells
- Features can be labeled to capture semantic information about the model



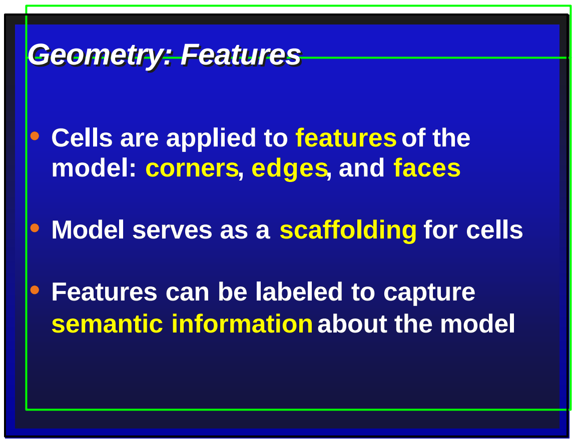
Geometry: Features

- Cells are applied to features of the model: corners, edges, and faces
- Model serves as a scaffolding for cells
- Features can be labeled to capture semantic information about the model



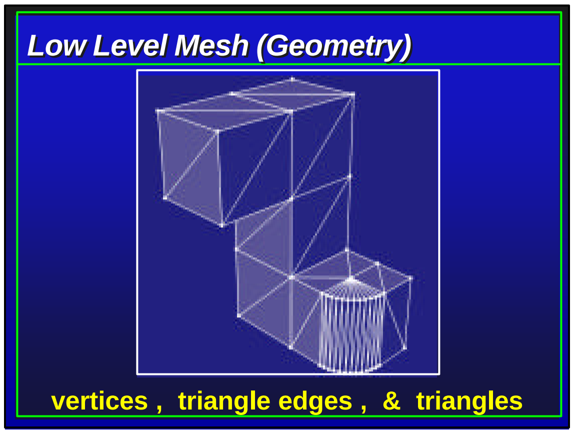
Geometry: Features

- Cells are applied to features of the model: corners, edges, and faces
- Model serves as a scaffolding for cells
- Features can be labeled to capture semantic information about the model

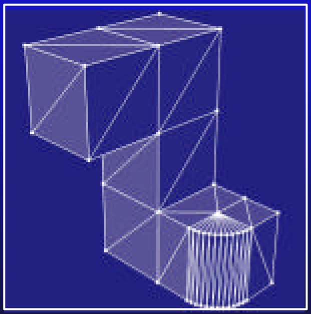


Geometry: Features

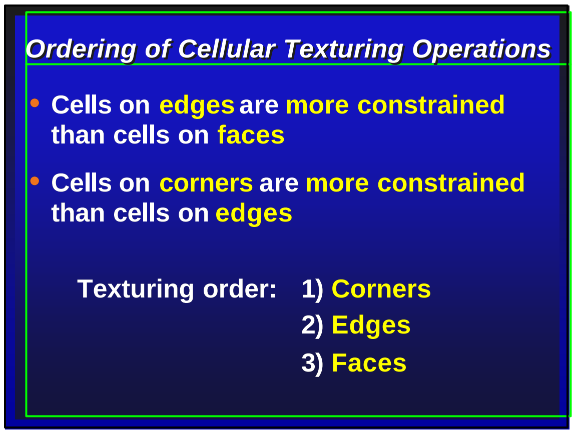
- Cells are applied to features of the model: corners, edges, and faces
- Model serves as a scaffolding for cells
- Features can be labeled to capture semantic information about the model



Low Level Mesh (Geometry)



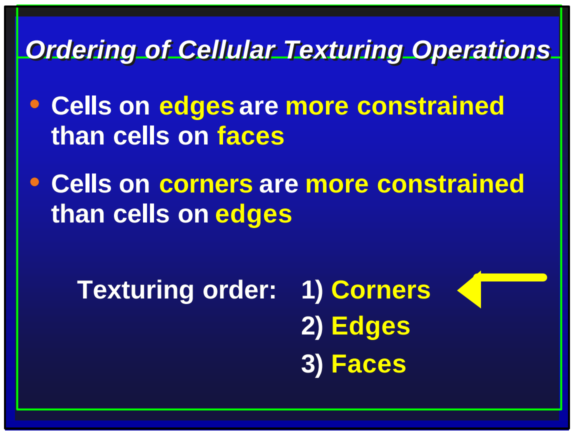
vertices , triangle edges , & triangles



Ordering of Cellular Texturing Operations

- Cells on edges are more constrained than cells on faces
- Cells on corners are more constrained than cells on edges

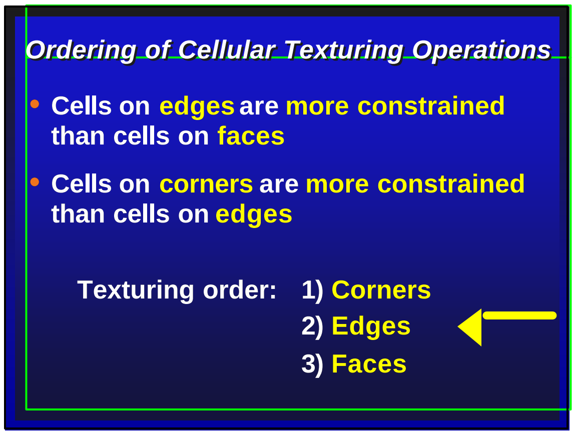
Texturing order: 1) Corners
2) Edges
3) Faces



Ordering of Cellular Texturing Operations

- Cells on edges are more constrained than cells on faces
- Cells on corners are more constrained than cells on edges

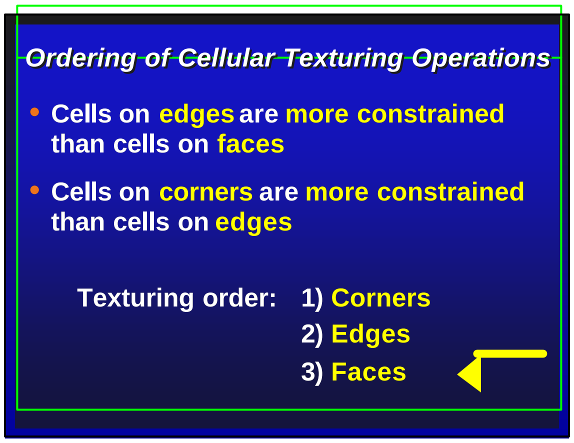
Texturing order: 1) Corners ←
2) Edges
3) Faces



Ordering of Cellular Texturing Operations

- Cells on edges are more constrained than cells on faces
- Cells on corners are more constrained than cells on edges

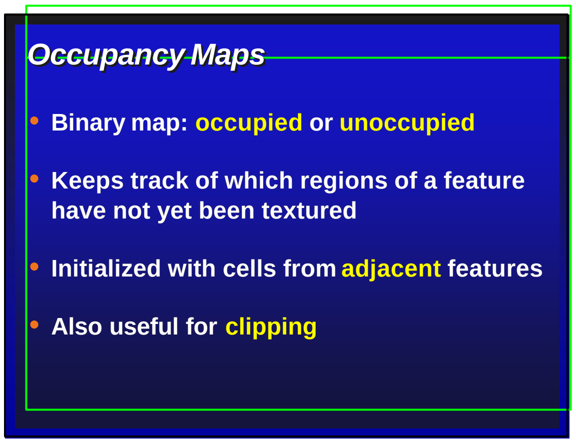
Texturing order: 1) Corners
2) Edges ←
3) Faces



Ordering of Cellular Texturing Operations

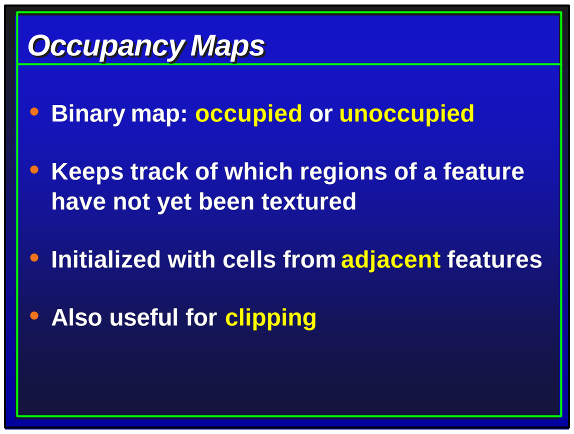
- Cells on edges are more constrained than cells on faces
- Cells on corners are more constrained than cells on edges

Texturing order: 1) Corners
2) Edges
3) Faces ←



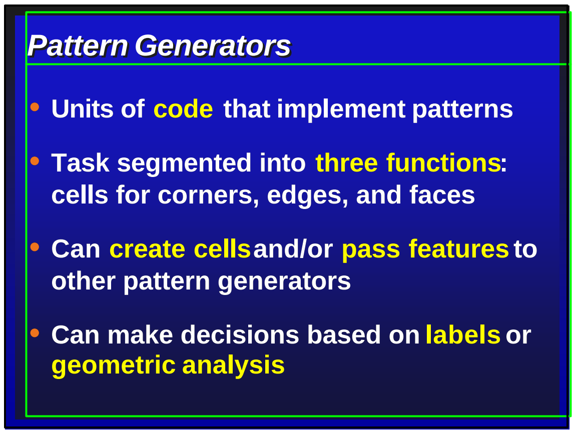
Occupancy Maps

- Binary map: occupied or unoccupied
- Keeps track of which regions of a feature have not yet been textured
- Initialized with cells from adjacent features
- Also useful for clipping



Occupancy Maps

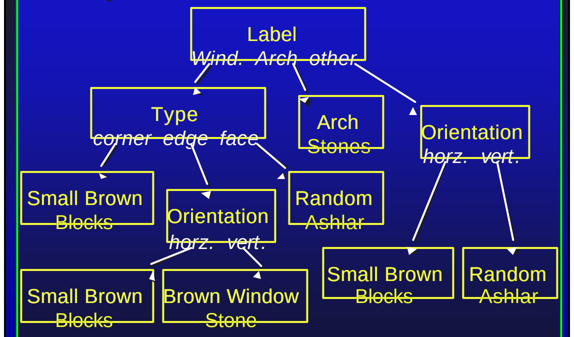
- Binary map: occupied or unoccupied
- Keeps track of which regions of a feature have not yet been textured
- Initialized with cells from adjacent features
- Also useful for clipping



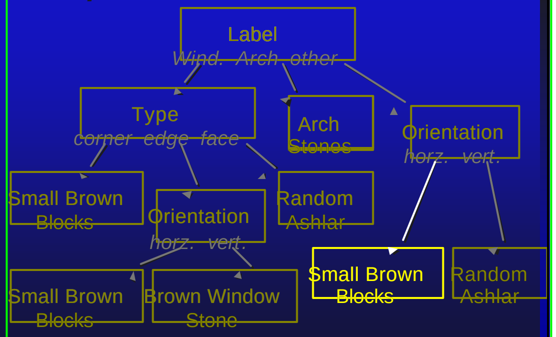
Pattern Generators

- Units of code that implement patterns
- Task segmented into three functions: cells for corners, edges, and faces
- Can create cells and/or pass features to other pattern generators
- Can make decisions based on labels or geometric analysis

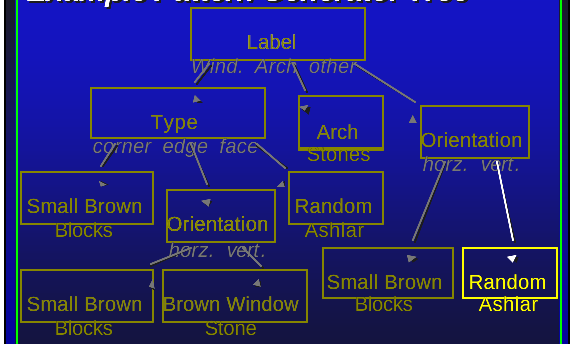
Example Pattern Generator Tree



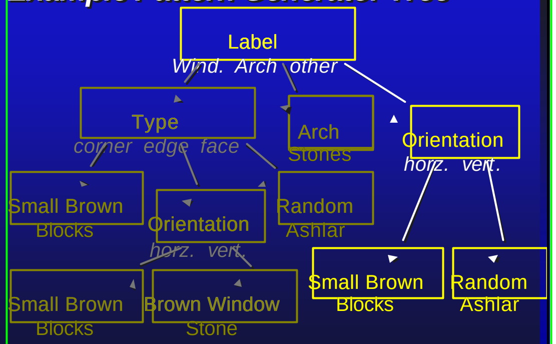
Example Pattern Generator Tree



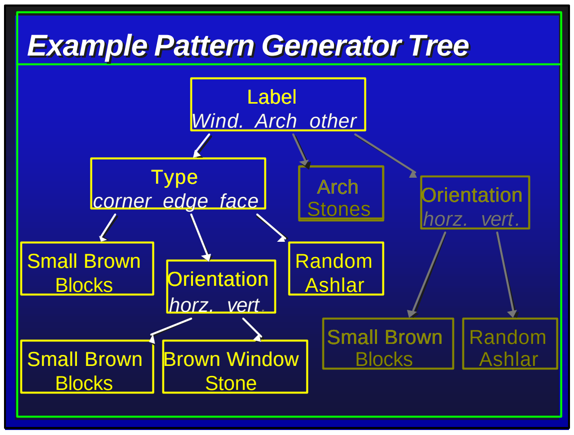
Example Pattern Generator Tree



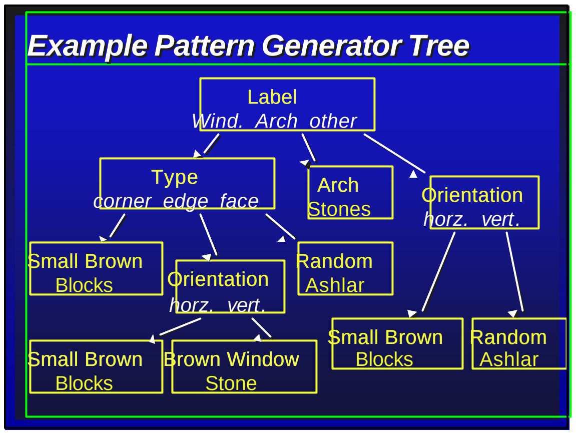
Example Pattern Generator Tree

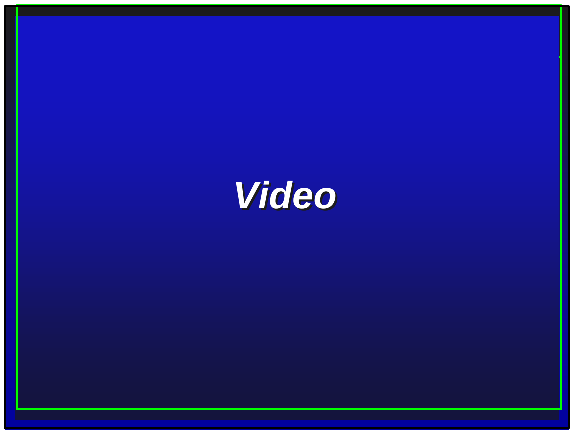


Example Pattern Generator Tree



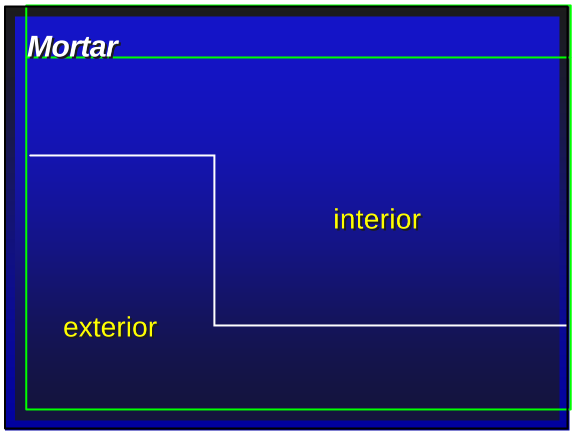
Example Pattern Generator Tree



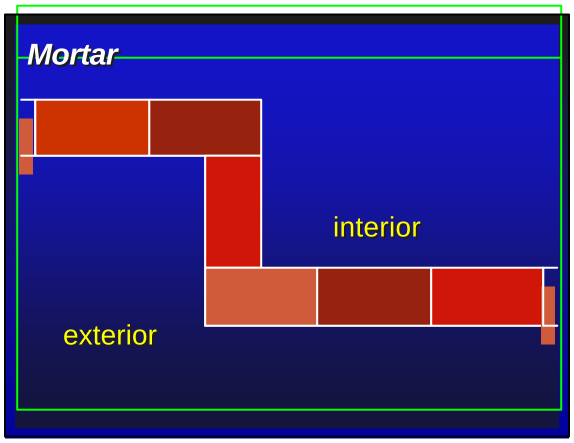


Video

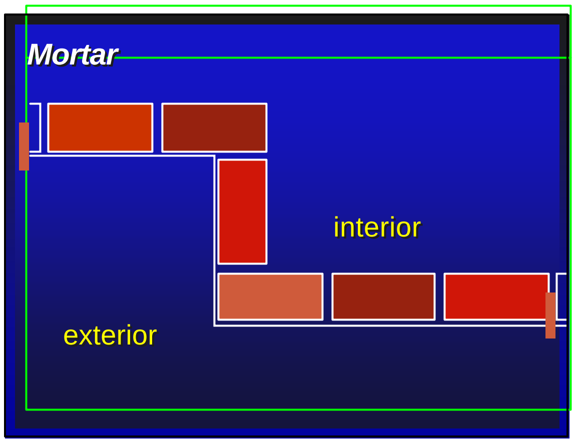
Mortar



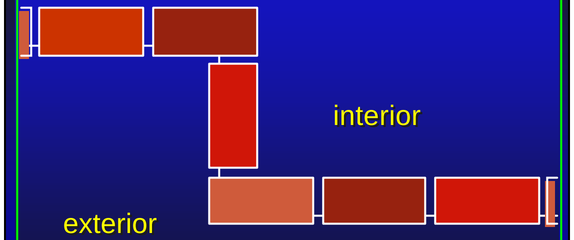
Mortar



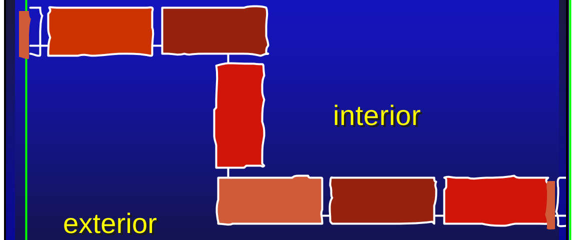
Mortar



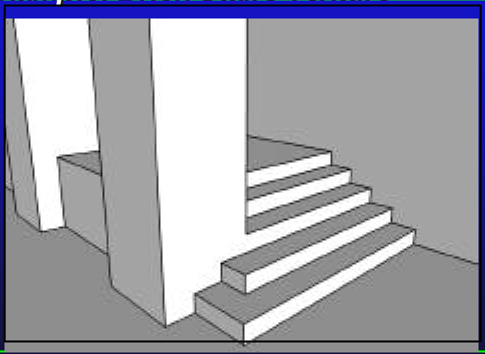
Mortar



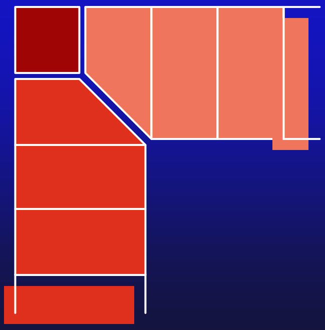
Mortar



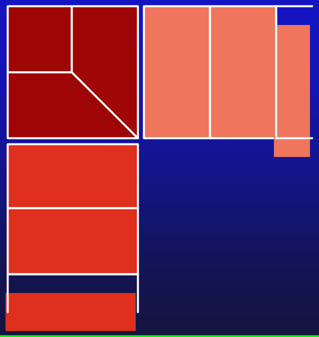
Example: Brick Stairs Texture



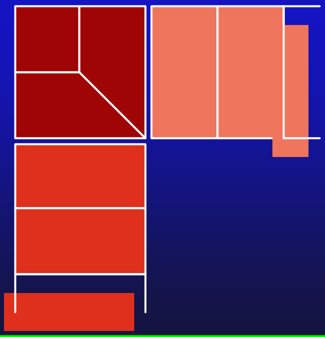
Pattern Segmentation

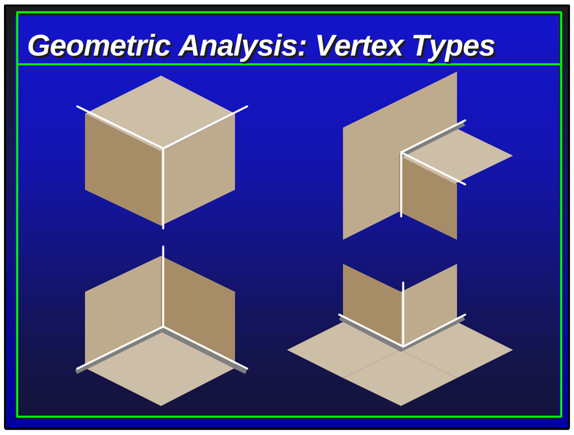


Pattern Segmentation

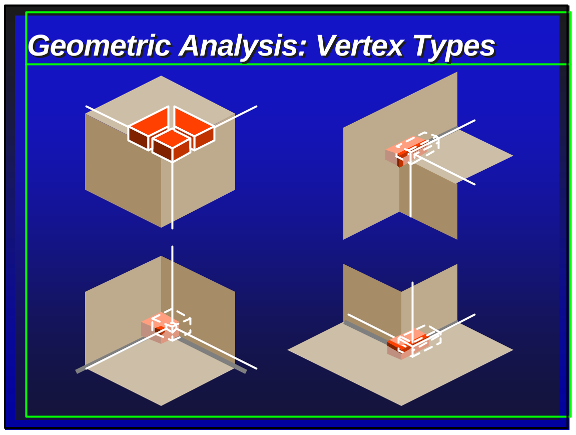


Pattern Segmentation

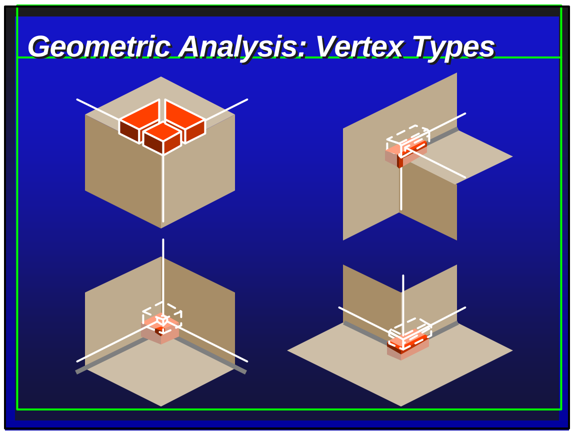




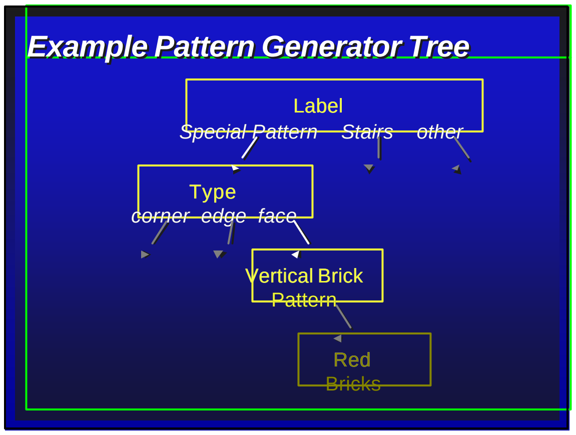
Geometric Analysis: Vertex Types



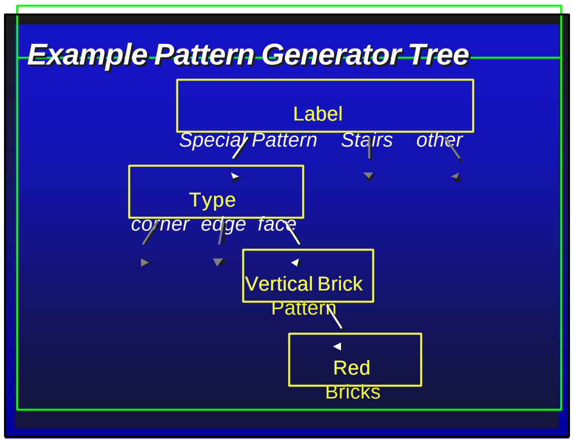
Geometric Analysis: Vertex Types



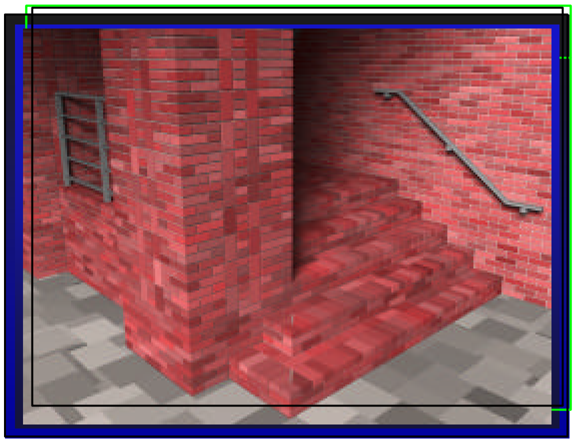
Geometric Analysis: Vertex Types

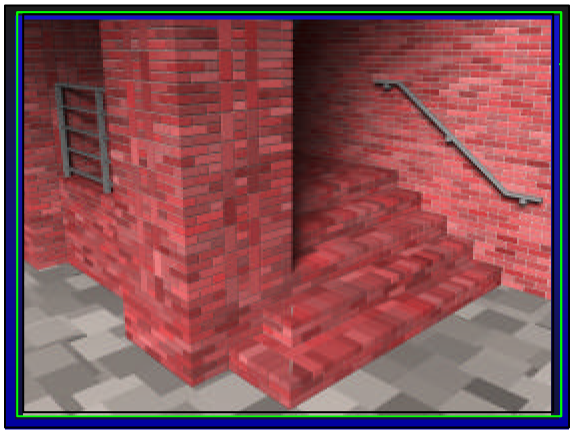


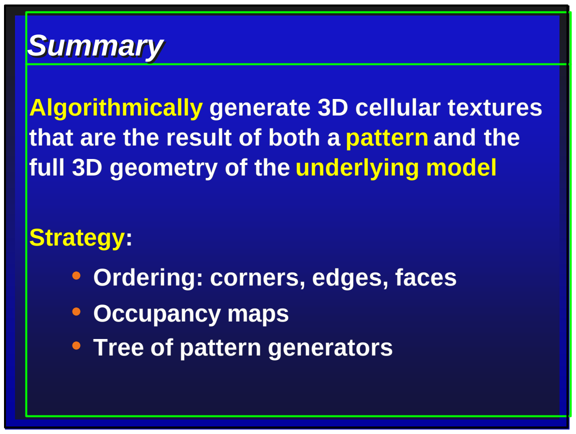
Example Pattern Generator Tree



Example Pattern Generator Tree





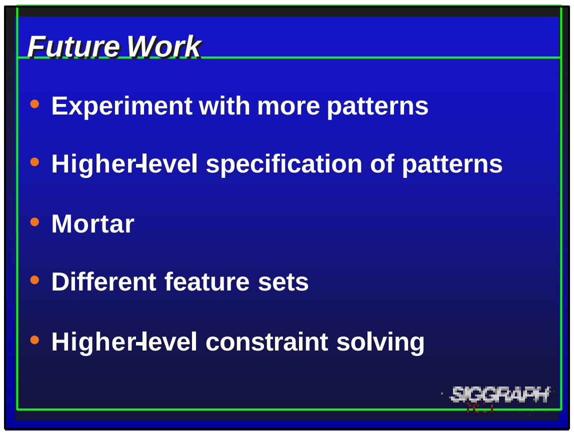


Summary

Algorithmically generate 3D cellular textures that are the result of both a pattern and the full 3D geometry of the underlying model

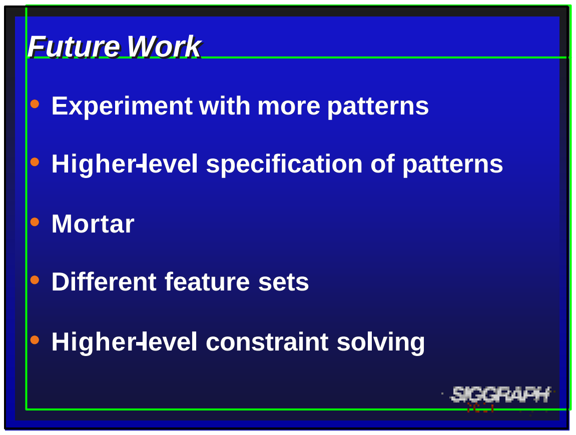
Strategy:

- **Ordering: corners, edges, faces**
- **Occupancy maps**
- **Tree of pattern generators**



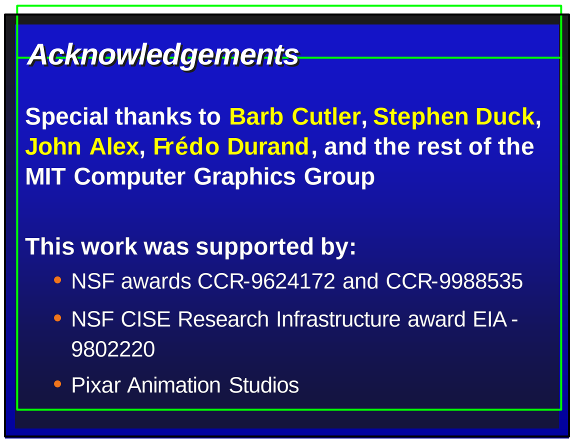
Future Work

- **Experiment with more patterns**
- **Higher-level specification of patterns**
- **Mortar**
- **Different feature sets**
- **Higher-level constraint solving**



Future Work

- **Experiment with more patterns**
- **Higher-level specification of patterns**
- **Mortar**
- **Different feature sets**
- **Higher-level constraint solving**



Acknowledgements

Special thanks to Barb Cutler, Stephen Duck, John Alex, Frédo Durand, and the rest of the MIT Computer Graphics Group

This work was supported by:

- NSF awards CCR-9624172 and CCR-9988535
- NSF CISE Research Infrastructure award EIA-9802220
- Pixar Animation Studios